RESEARCH REPORT SERIES
(Disclosure Avoidance #2018-02)

Rosetta Wiki 1.0 User Guide:

Companion to Rosetta Wiki

Nelson Chung, Steve Clark, Philip Leclerc, Aref Dajani and Phyllis Singer

Center for Disclosure Avoidance Research
U.S. Census Bureau
Washington DC 20233

Report Issued: September 2018

Table of Contents

1 Introduction

This document introduces the content and the motivation behind the U. S. Census Bureau Rosetta Wiki. It directs the reader to the Wiki's online location, provides an example on how the Wiki is used, and contains instructions on expanding the Wiki. In addition to the main content, appendices guide the reader on using the code editor RStudio, which the primary author has offered for R instruction at the Center for Disclosure Avoidance Research (CDAR) and has made available a hard-copy version of the Wiki code. There are no Census Bureau data presented in this user guide.

This user's guide is geared towards both new and proficient statisticians and programmers. As a result, you will find material presented at various skill levels. If you have any questions, feel free to contact the first author, Nelson Chung, at x3-3490.

2 What is the Rosetta Wiki?

There are several major ways to learn software; among them are reading documentation, attending a class where one observers as an instructor, inheriting code in a language one does not know, and being coached by a mentor or colleague one-one-one. The former often entails a steeper learning curve. The latter only enables the student to learn specific tasks, without the theoretical knowledge necessary to solve problems when unforeseen roadblocks appear. The Rosetta Wiki offers what works best from each method; it is a repository of task-driven code designed to enable programmers from diverse backgrounds, such as software engineering, data analysis, statistics, and applied mathematics with experience in at least one of the available languages to translate code to accomplish their task from a known coding language into corresponding code for accomplishing their task in a coding language they do not yet know. The Rosetta Wiki currently contains code in three languages: R, Python (version 3.4), and SAS.

3 Related Work

Mike Mol also created a repository of tasks in over 650 different programming languages, ranging from Ada to Ruby, called Rosetta Code (2007). This Rosetta Wiki will focus on tasks of primary interest to the U.S. Census Bureau; i.e., writing statistical and modeling software, and is geared towards Census employees. This assumes that work is conducted behind the U. S. Census Bureau firewall in the presence of the usual IT restrictions on program installation. Helper programs, modules, and libraries available to Census employees are present in the user's system PATH.

Books for translating between statistical programming languages have previously been authored (Klein and Horton 2014; Muenchen 2016; Ohri 2017). This Rosetta Wiki is a similar, interactive resource that is not only open-source, but also easier to implement.

4 Motivation

The Rosetta Wiki grew out of the need for statisticians at CDAR to transition from SAS to R and Python. The Wiki's code accompanied the lecture notes for R training. It can also serve as a tool for new hires, proficient typically in R or Python, to learn SAS in order to communicate with the U. S. Census Bureau's more seasoned staff.

5 Authors

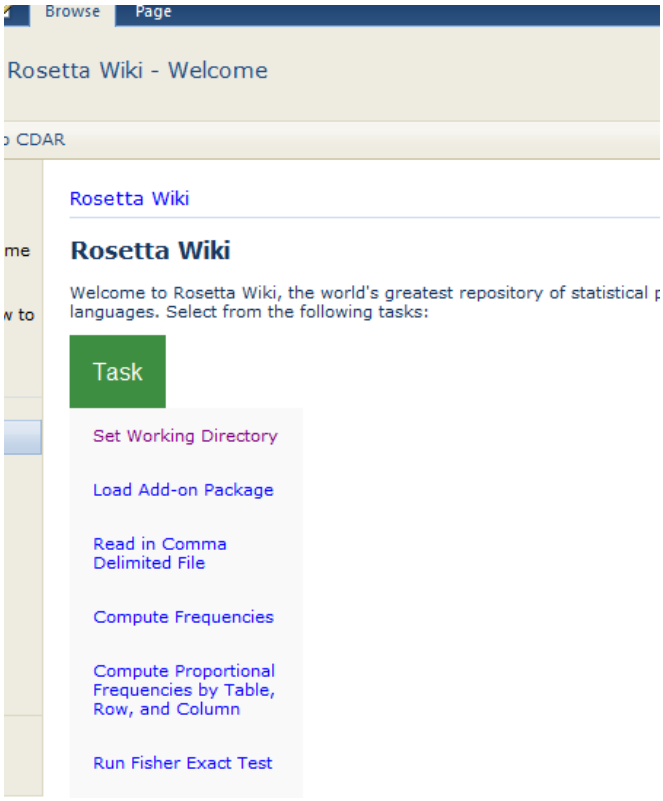
Code for the Rosetta Wiki was written by the authors of this document, under the general direction of Aref Dajani. Nelson Chung was primarily responsible for the R code, Philip Leclerc for the Python code, Steve Clark and Phyllis Singer for the SAS code. The Wiki was beta-tested by several mathematical statisticians in CDAR. To ensure objectivity, each statistician was permitted to test code in the same module (task), but not code that she wrote herself.

6 Using the Rosetta Wiki

Because we anticipate more users than contributors for the Wiki, we use the Wiki for its main purpose: namely to perform programming tasks. First, we explain the structure of the Wiki. Then we discuss how to access and use the Wiki.

6.1 Structure of the Rosetta Wiki. As mentioned earlier, the Rosetta Wiki is housed at the CDAR Sharepoint site. The edit-ability of Sharepoint enables Wiki functionality. Code for each programming task is found as individual pages with its own web address. All pages are accessible from the Rosetta Wiki welcome page, by moving the mouse over the blue "Task" button that turns green, shown in Figure 6.1.1:

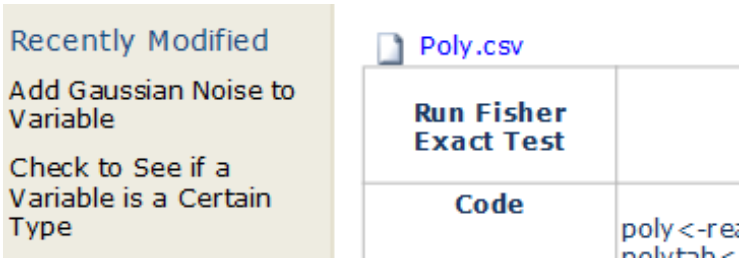
Figure 6.1.1 Rosetta Wiki Welcome Page



6.2 Locating the Rosetta Wiki. For U. S. Census Bureau employees, the Rosetta Wiki is found on the CDAR SharePoint page at: https://collab.ecm.census.gov/div/cdar/SitePages/Home.aspx. Click on "Training" in the bar on top, and that should take the coder to "CDAR Training Links." The first link is "Rosetta Wiki – Welcome." There, the coder will find a blue drop-down button that turns green with the tasks listed when she hovers over it. The code is there for the programmer to copy and paste into an Integrated Development Environment to run.

6.3 Example of Using the Wiki. We now present an example of how the users familiar with a programming task in one language can use the Wiki to program the same task in another language. This illustration applies when, say a supervisor in CDAR asks a recent hire who knows Python but not SAS to run a Fisher Exact test, and formulate the results in SAS. The Fisher test is an F-test, and the Fisher Exact test is for small sample sizes (In fact, the F-test is Fisher's namesake). We go on the Sharepoint site, mouse-over the "Task" button and choose "Run Fisher Exact Test." As can be seen in Figure 6.2.1, there is a link to the comma delimited file, Poly.csv. The left, platinum-background sidebar lists web pages that were recently modified. They may be internal or external to this Rosetta Wiki.

Figure 6.2.1 Upper Left-hand Corner of Fisher Exact Test Code Page



Through this Rosetta Wiki, the coder is able to access the familiar Python code for running the Fisher exact test. As a preliminary note, we will denote all variable names found in the prose with SMALL CAPS. The code first imports Poly.csv, a dataset of public information of college football players with columns NAME, HERITAGE, and STATUS, from the working directory. The test will be used to determine whether there is a statistically significant difference in STATUS (Missed or Signed) between two groups that vary by HERITAGE, Polynesian or Non-Polynesian.

Table 1. MS Excel Layout of First Ten Rows of Poly.csv[1]

Position	Name	Heritage	Status
DT	Haloti Ngata	Polynesian	Missed
DE	J. T. Mapu	Polynesian	Missed
OL	Ryan Carter	Non-Polynesian	Missed
OL	Tautusi Lutui	Polynesian	Missed
DE	C. J. Ah You	Polynesian	Missed
LB	Kaluka Maiava	Polynesian	Missed
OL	Adam Hawes	Non-Polynesian	Missed
OL	Fenuki Tupou	Polynesian	Missed
DT	Sione Fua	Polynesian	Missed
DT	Sealver Siliga	Polynesian	Missed

Figure 6.3.1 is a partial screenshot of the Sharepoint page. It displays the widest area of the page subject to the constraints of this document. In the upper-left hand quadrant is the Python code to run a Fisher

[1] None of these data are confidential.

exact test; in the lower-left hand quadrant, the output. In the upper-right hand quadrant is the SAS code for the same test; in the lower right-hand quadrant, the SAS output.

Figure 6.3.1 Fisher Exact Test Sharepoint Page

Run Fisher Exact Test

R

Code

```
poly<-read.csv("Poly.csv")
polytab<-table(poly$Status,poly$Heritage)
print(polytab)
fish<-fisher.test(polytab,alternative="less",conf.int=F)
print(noquote(sprintf('%.4g',fish$p.value)))
```

Desired Result

```
        Non-Polynesian Polynesian
  Missed              4         21
  Signed              8          9
[1] 0.03342
```

Python

Code

```
import pandas as pd
import scipy.stats as stats
polytab=pd.crosstab(poly.Status,poly.Heritage)
print(polytab)
oddratio,pvalue=stats.fisher_exact(polytab,alternative='less')
print(format(pvalue,'.4g'))
```

Desired Result

```
Heritage  Non-Polynesian  Polynesian
Status
Missed               4         21
Signed               8          9
0.03342
```

SAS

Code

```
proc import datafile="poly.csv" out=poly dbms=csv replace;
proc freq data=poly;
 tables Status*Heritage /norow nocol nopercent;
run;

proc freq noprint data=poly;
  tables Status*Heritage /chisq;
  output out=poly2(keep=xpl_fish) chisq;
run;

proc print data=poly2;
  title 'Fisher Exact Left-Sided P-value';
run;
```

Desired Result

Frequency

Table of Status by Heritage

Status	Heritage		
	Non-Polynesian	**Polynesian**	**Total**
Missed	4	21	25
Signed	8	9	17
Total	12	30	42

Obs	XPL_FISH
1	0.0334

When we zoom in to the third column quadrant where the Python code is housed, we observe the following, (explanations added to the comments following a # sign that denotes comments):

```
import pandas as pd

import scipy.stats as stats
polytab=pd.crosstab(poly.Status,poly.Heritage)
print(polytab)
oddratio,pvalue=stats.fisher_exact(polytab,alternative='less')
print(format(pvalue,'.4g'))
```

The code dictates the program to print results of the cross-tabulation and the p-value. The user can copy the above code and it paste it into an Interactive Development Environment (IDE), yielding the following output. The yellow highlight for the p-value was added for purposes of this document.

```
Heritage  Non-Polynesian  Polynesian
Status
Missed                4          21
Signed                8           9
0.03342
```

The task is done in the following three steps. First, the Poly.csv file is read using the read_csv() function which is part of the pandas module. Pandas is a Python module for statistical analysis, and an acronym for "panel data." We can designate it with the shorthand "pd" in order to avoid having to repeatedly typing out the entire word. Second, a two-way frequency table is created using the crosstab() function, also of the pandas module. Finally, the fisher_exact() function from Python's stats module is used to determine whether or not the null hypothesis that STATUS and HERITAGE are the same, in effect, to determine whether observations that belonged to one HERITAGE group were less likely to have a certain STATUS as the other HERITAGE group.

The crux of the task is executed in the second-to-last line. A one-sided Fisher test is performed, with the two HERITAGE groups being the same with regards to STATUS as the null hypothesis, and the Non-Polynesian group being less likely to sign than the Polynesian group as the alternative hypothesis. The function stats.fisher_exact outputs two values, the odds ratio and the p-value, which we designate as oddratio and pvalue. Then direct Python to print the p-value, set to four significant digits.

To implement those three steps in SAS, we move to the fourth column, giving us the following code:

```
proc import datafile="poly.csv" out=poly dbms=csv replace;
options nocenter pageno=1;
ods pdf file='M:\Rosetta Wiki\SAS\fisher_exact.pdf';
ods pdf startpage=no;

proc import datafile="poly.csv" out=poly dbms=csv replace;
proc freq data=poly;
 tables Status*Heritage /norow nocol nopercent;
run;
proc freq noprint data=poly;
  tables Status*Heritage /chisq;
  output out=poly2(keep=xpl_fish) chisq;
run;

proc print data=poly2;
  title 'Fisher Exact Left-Sided P-value';
run;
```

The coder imports the CSV file Poly.csv and outputs it into a SAS dataset called “poly,” and designates the database management system (DBMS) as “csv” for a CSV file. The “replace” directs the SAS engine to have the new dataset replace any datasets created during the session named “poly.” Frequencies are computed, then a chi-square test, with extra code to suppress all the default output that we do not need so that all we see is the p-value for the Fisher Exact test. The code produces the output as was seen in Figure 6.3.1.

7 Next Steps

It is intended that code for more tasks and additional languages will be added as the CDAR’s needs evolve, including legacy and innovative disclosure avoidance methods and techniques. In this Rosetta Wiki interface, code for specific tasks is presented side-by-side. In this user’s guide, code and output will be presented “stacked”; i.e., the code and output will be presented sequentially and not in parallel. The ultimate goal is to make this Wiki query-able, isolating two languages in the user interface like Google Translate, enabling the user to enter a task in one language and have it show up in one other.

8 Conclusion

In this paper, we introduced the Rosetta Wiki, a repository of task-driven code in R, Python, and SAS, situated this Wiki among related previous work, unpacked the motivation behind it, and discussed how to use it with examples. This paper contains instructions on augmenting the Wiki, enabling further work to be done.

9 References

CRAN. The Comprehensive R Archive Network. https://cran.r-project.org

Google. Google Translate. https://translate.google.com

Kleinman, Ken and Nicholas J. Horton. 20D. SAS and R. Boca Raton, FL: CRC Press.

Mol, Mike. 2007. Rosetta Code. http://rosettacode.org/wiki/Rosetta_Code.

Muenchen, Robert A. 2016. R for SAS and SPSS Users. New York: Springer.

Ohri, Ajay. 2017. Python for R Users. New York: Wiley.

Python. The Official Home of the Python Programming Language. https://www.python.org

SAS. SAS: Analytics, Business Intelligence, and Data Management.
https://www.sas.com/en_us/home.html

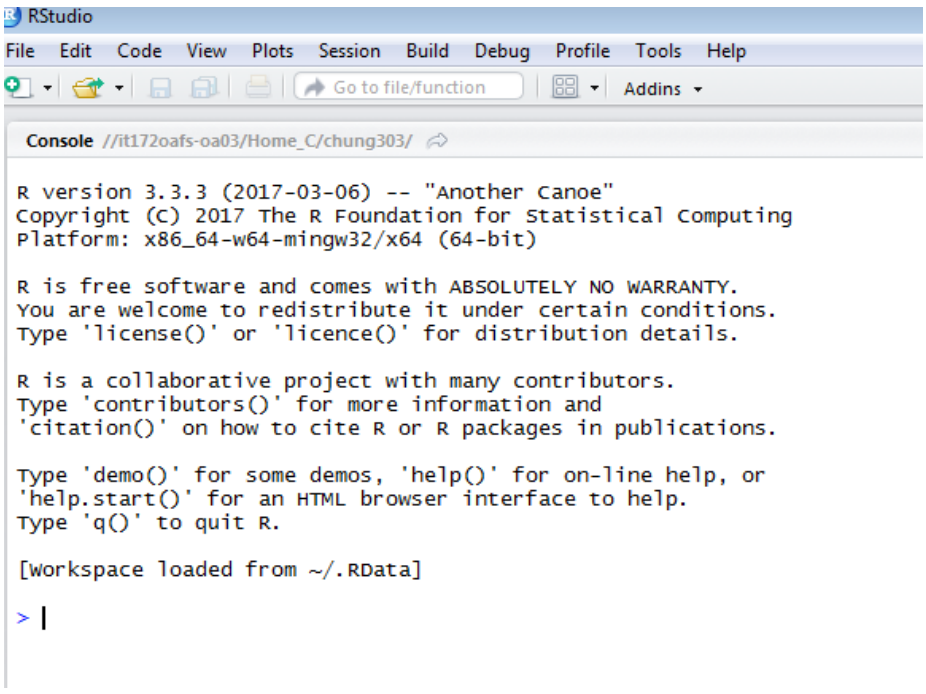
Appendix A: RStudio (for new R programmers)

For code that is relatively long, coding may be better done in a text editor or interactive development environment (IDE). In this section, we recommend using RStudio for programming in R. We chose RStudio because it is available at the U.S. Census Bureau. When outside the Bureau, or not using its machines, this can be downloaded from http://rstudio.com, with necessary instructions on installation. After download, click on the R icon from your desktop, which should look like Figure A.1:

Figure A.1 RStudio Icon

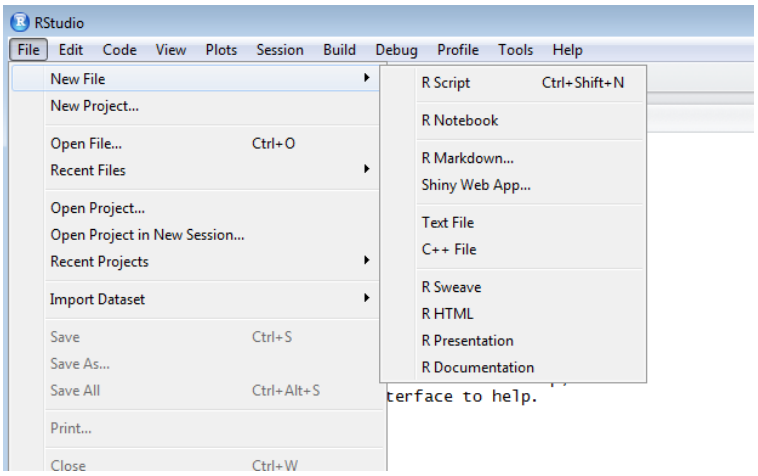
When the coder opens R Studio, she should see something akin to Figure A.2:

Figure A.2 R Start Screen



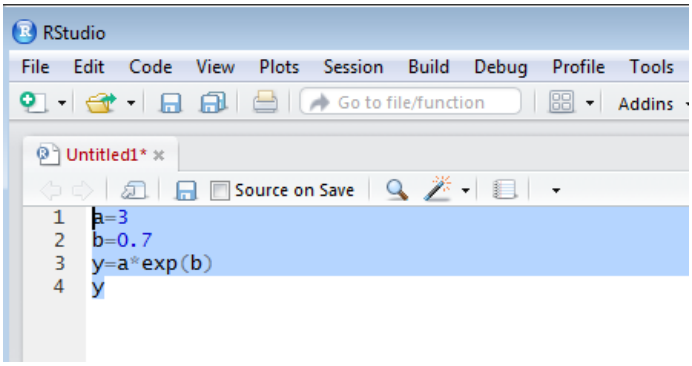
There are many useful features in R Studio. Explore the entire application. To focus on the left half of the interactive environment, which is the R console, here is what one would see as shown in Figure A.3.

Figure A.3 Opening R Script



We recommend a text editor for longer code. R Script, R Notebook, or R Markdown are found by selecting "File," then "New File." R Script is the most user-friendly of the three. Try entering and running code in the console, as shown in Figure A.4:

Figure A.4 R Console



Highlight the code, then click "Run." Alternatively, Ctrl+Enter runs the highlighted block. If no block is highlighted, Ctrl+Enter runs the line the cursor is on. Finally, click on the expand item on the console to see the result, which should appear in the console, as shown below in Figure A.5:

Figure A.5 R Function Code

```
> y
[1] 6.041258
> a=3
> b=0.7
> y=a*exp(b)
> y
[1] 6.041258
>
```

Appendix B: Getting Started (for new programmers)

The following is a demonstration of performing the following tasks in R, Python, and SAS using Linux and PC: opening a session, running a "Hello World!" program, closing the program. The first step is to write the program in a text editor. The availability of the three software programs variables by division. What we present is what is available in the current environment at CDAR. For most Census computers, SAS is pre-installed to the PC. R and Python can be installed after approval from the Census Standards Working Group.

B.1 Opening a Hello World Session

B.1.1 Opening a Hello World Session in Linux

To open R, Python, or SAS in the Linux environment, one should enter the following command to run the following at the command prompt, as shown in Figure B.1: `qsub -I -X`

Figure B.1 Opening Statistical Programs from Linux

```
[chung303@hpc-login2 ~]$ qsub -I -X
```

From there, the coder can run commands to open Hello world programs. The command can be found in the following table:

	Script File	Command
R	hello_world.R	Rscript hello_world.R
Python	hello_world.py	python hello_world.py
SAS	hello_world.sas	qsas hello_world

B.1.2 Opening a Hello World Program in PC

SAS, and R and Python (after they are installed) are accessible from the start menu. See Figures B.2-B.4.

Figure B.2 Opening PC R

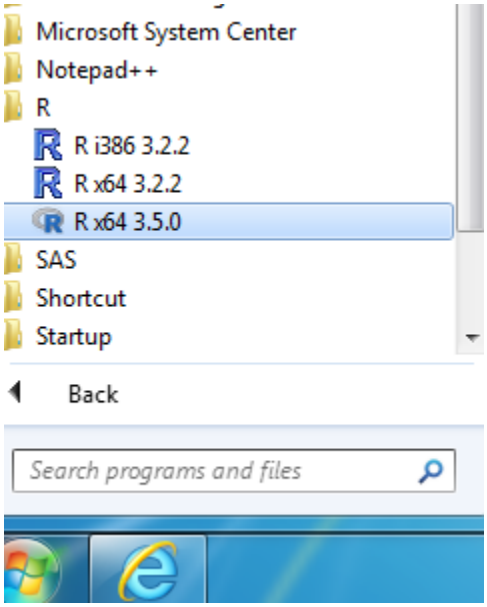


Figure B.3 Opening PC Python

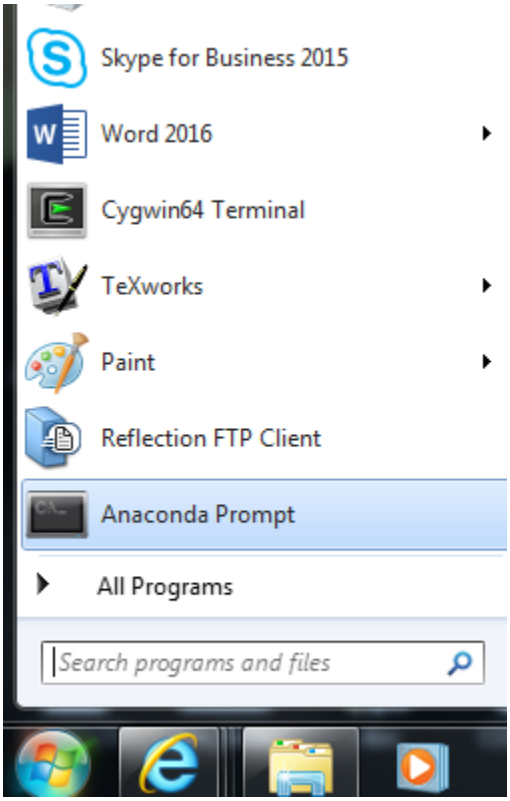
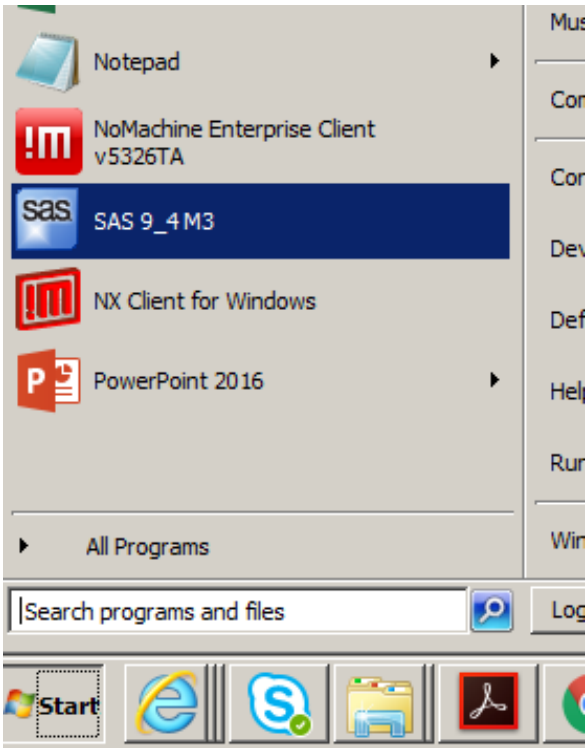


Figure B.4 Opening PC SAS



B.2 Running a Hello World Program

The actual script for the Hello World program for which instructions are found in section B.1 in the three languages and their accompanying results are:

	Code	Desired Result
R	print("Hello World!", quote = F)	[1] Hello World!
Python	print("Hello World!")	Hello World!
SAS	data hello_world; Hello_World="Hello World!"; run; proc print data=hello_world; var Hello_World; run;	Obs Hello_World 1 Hello World!

B.3 Terminate a Session

	Code
R	quit()
Python	exit()
SAS	endsas;

Appendix C: Beginning Functions (for new programmers)

C.1 Set Working Directory and Return Working Directory

The following task:

1. sets the working directory, enabling the user to avoid typing out the entire path each time, and
2. print the working directory to ensure it was done properly.

The print() functions in R and Python are necessary to print whenever we run a saved script. When just running code from within the environment, the function is not necessary; i.e., instead of print(getwd()) or print(os.getcwd()), just getwd() or getcwd(), respectively, will do.

Set Working Directory	Code	Desired Result
R	setwd('M:/Users/Nelson/Rosetta Wiki/' print(getwd()))	[1] "M:/Users/Nelson/Rosetta Wiki"
Python	os.chdir("M:/Users/Nelson/Rosetta Wiki/Python/") print(os.getcwd())	'M:\\Users\\Nelson\\Rosetta Wiki\\Python'
SAS	libname rosetta 'M:\Users\Nelson\Rosetta Wiki\SAS\';	1 libname rosetta 'M:\Users\Nelson\Rosetta Wiki\SAS ;NOTE: Libref ROSETTA was successfully assigned as follows: Engine: V9 Physical Name: M:\Users\Nelson\Rosetta Wiki\SAS

C.2 Install and Load Add-On Package

The following task entails installing and loading an add-on package. The R console lists the dependencies required for the package. One may wonder what the dependency *zoo* does; it contains functions for time series, often necessary for computing stochastic differential equations. This particular R package, *sde*, contains a description with an unusual amount of detail. This function is not readily available in SAS.

Load Add-on Package	Code	Desired Result
R	setwd('M:/Users/Nelson/Rosetta Wiki/') print(getwd())	**To install:** install.packages("sde") **To Load:** library(sde) Loading required package: MASS Loading required package: stats4 Loading required package: fda Loading required package: splines Attaching package: 'fda' The following object is masked from 'package:graphics': matplot Loading required package: zoo Attaching package: 'zoo' The following objects are masked from 'package:base': as.Date, as.Date.numeric sde 2.0.15 Companion package to the book 'Simulation and Inference for Stochastic Differential Equations With R Examples' Iacus, Springer NY, (2008) To check the errata corrige of the book, type vignette("sde.errata")
Python	**Note: The following code should be entered after opening up Ananconda and prior to entering Python, *when the coder is not at the U. S. Census Bureau.*** **To install:** pip install sdeint **To Load (In Python):** import sdeint 1 l	
SAS	**Not readily available**	

C.3 Concatenate Matrices

In this module, matrices are created and joined together. Unlike Python and SAS, R fills matrices along columns instead of rows. Therefore, it's necessary to use the byrow=T argument with matrix(). Python output is stored in .txt files because its display of matrices in double brackets creates problems in HTML. In C.3.1, two 2 by 4 matrices are created. In C.3.2, those two are joined horizontally. In C.3.2, those two are joined vertically. For SAS, `proc iml` requires that IML is installed. Note that Python requires the modules *numpy* for numerical processing, and *pandas*, which transforms the syntax to resemble R to a remarkable degree.

C.3.1 Create a 2x4 Matrix

Create 2X4 Matrix	Code	Desired Result
R	install.packages("abind") library(abind) a<-matrix(1:8,2,4,byrow=T) print(a) b<-matrix(9:16,2,4,byrow=T) print(b)	[,1] [,2] [,3] [,4] [1,] 1 2 3 4 [2,] 5 6 7 8 [,1] [,2] [,3] [,4] [1,] 9 10 11 12 [2,] 13 14 15 16

Python	import numpy as np
	
	a = np.array(list(range(1,9))).reshape(2,4)
	print(a)
	b = np.array(list(range(9,17))).reshape(2,4)
	print(b)
	
	ab = np.concatenate((a,b),axis=1) # Join two matrices column-wise
	print(ab)
	
	c = np.array(list(range(1,25))).reshape(6,4)
	print(c)
	ac = np.concatenate((a,c),axis=0) # Join two matrices row-wise
	print(ac)
		[[1 2 3 4]
		[5 6 7 8]]
		
		[[9 10 11 12]
		[13 14 15 16]]

SAS	proc iml;
	x=1:8;
	a=shape(x,2,4);
	print(a);
	y=9:16;
	b=shape(y,2,4);
	print(b);
	quit;
		a
		1 2 3 4
		5 6 7 8
		
		b
		9 10 11 12
		13 14 15 16

C.3.2 Join Matrices Horizontally

Join Matrices Horizontally	Code	Desired Result

R	#Simlar to cbind for dataframes
	ab=abind(a,b,along=2)
	print(ab)
		[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
		[1,] 1 2 3 4 9 10 11 12
		[2,] 5 6 7 8 13 14 15 16

Python	ab = np.concatenate((a,b),axis=1) print(ab)	[[1 2 3 4 9 10 11 12] [5 6 7 8 13 14 15 16]]

SAS	proc iml;
	x=1:8;
	a=shape(x,2,4);
	print(a);
	y=9:16;
	b=shape(y,2,4);
	print(b);
	ab=a||b;
	print(ab);
	quit;
		ab
		1 2 3 4 9 10 11 12
		5 6 7 8 13 14 15 16

The Python tasks requires the numpy module (abbreviated in the code as np) to form an array. In both R and Python, arrays are objects that contain elements of a single type.

C.3.3 Join Matrices Vertically

Join Matrices Vertically	Code	Desired Result

R	c<-matrix(1:24,6,4,byrow=T)
	print(c)
	
	#Join two matrices vertically
	
	#Similar to rbind() for dataframes
	ac=abind(a,c,along=1)
	print(ac)
		[,1] [,2] [,3] [,4]
		[1,] 1 2 3 4
		[2,] 5 6 7 8
		[3,] 9 10 11 12
		[4,] 13 14 15 16
		[5,] 17 18 19 20
		[6,] 21 22 23 24
		
		[,1] [,2] [,3] [,4]
		[1,] 1 2 3 4
		[2,] 5 6 7 8
		[3,] 1 2 3 4
		[4,] 5 6 7 8
		[5,] 9 10 11 12
		[6,] 13 14 15 16
		[7,] 17 18 19 20
		[8,] 21 22 23 24

Python	c = np.array(list(range(1,25))).reshape(6,4)
	print(c)
	ac = np.concatenate((a,c),axis=0) # Join two matrices row-wise
	print(ac)
		[[1 2 3 4]
		[5 6 7 8]
		[9 10 11 12]
		[13 14 15 16]
		[17 18 19 20]
		[21 22 23 24]]
		
		[[1 2 3 4]
		[5 6 7 8]
		[1 2 3 4]
		[5 6 7 8]
		[9 10 11 12]
		[13 14 15 16]
		[17 18 19 20]
		[21 22 23 24]]

SAS	proc iml;
	x=1:8;
	a=shape(x,2,4);
	print(a);
	y=9:16;
	b=shape(y,2,4);
	print(b);
	ab=a||b;
	print(ab);
	z=1:24;
	c=shape(z,6,4);
	print(c);
	ac=a//c;
	print(ac);
	quit;
		c
		1 2 3 4
		5 6 7 8
		9 10 11 12
		13 14 15 16
		17 18 19 20
		21 22 23 24
		
		ac
		1 2 3 4
		5 6 7 8
		1 2 3 4
		5 6 7 8
		9 10 11 12
		13 14 15 16
		17 18 19 20
		21 22 23 24

C.4 Matrix Mathematical Operations

The following code performs matrix transpose, matrix multiplication, and matrix inversion.

C.4.1 Transpose a Matrix

Transpose a Matrix	Code	Desired Result
R	install.packages("optimbase") install.packages("abind") library(optimbase) library(abind) a=matrix(1:4,2,2,byrow = T) print(a) at=transpose(a) print(at)	[,1] [,2] [1,] 1 2 [2,] 3 4 [,1] [,2] [1,] 1 3 [2,] 2 4
Python	a=np.array(list(range(1,5))).reshape(2,2) print(a) at=a.transpose() print(at)	[[1 2] [3 4]] [[1 3] [2 4]]

SAS	proc iml;
	x=1:4;
	a=shape(x,2,2);
	print(a);
	at=t(a);
	print(at);
	quit;
		a
		1 2
		3 4
		
		at
		1 3
		2 4

C.4.2 Multiply Matrices

Multiply Matrices	Code	Desired Result
R	a<-matrix(c(5,2,7,3),2,2,byrow=T) print(a) a_inv=solve(a) print(a_inv) b=matrix(c(3,5,2,2,9,8),3,2,byrow=T) print(b) print(b%*%a)	[,1] [,2] [1,] 5 2 [2,] 7 3 [,1] [,2] [1,] 3 5 [2,] 2 2 [3,] 9 8 [,1] [,2] [1,] 50 21 [2,] 24 10 [3,] 101 42
Python	import numpy as np a=np.matrix('5 2; 7 3') print(a) b=np.matrix('3 5; 2 2; 9 8') print(b) print(np.matmul(b,a))	[[5 2] [7 3]] [[3 5] [2 2] [9 8]] [[50 21] [24 10] [101 42]]

SAS	proc iml;
	a={5 2, 7 3};
	print(a);
	b={3 5,2 2, 9 8};
	print(b);
	ba=b*a;
	print(ba);
	quit;
		a
		5 2
		7 3
		
		b
		3 5
		2 2
		9 8
		
		ba
		50 21
		24 10
		101 42

C.4.3 Invert a Matrix

Take the Inverse of a Matrix	Code	Desired Result
R	a<-matrix(c(5,2,7,3),2,2,byrow=T) print(a) a_inv=solve(a) print(a_inv)	[,1] [,2] [1,] 5 2 [2,] 7 3 [,1] [,2] [1,] 3 -2 [2,] -7 5
Python	import numpy as np a=np.matrix('5 2; 7 3') print(a) a_inv=a.I print(a_inv)	 [[5 2] [7 3]] [[3. -2.] [-7. 5.]]

SAS	proc iml;
	a={5 2, 7 3};
	print(a);
	a_inv=inv(a);
	print(a_inv);
	quit;
		a
		5 2
		7 3
		
		a_inv
		3 -2
		-7 5

C.5 Merge Datasets

The following code imports two CSV files on players from the World Baseball Classic: wbc_team and wbc_stat and merges them by the key variable COUNTRY. The resulting dataset is extensive enough that we store them in separate text files.

Merge Datasets	Code	Desired Result
R	wbc_team<-read.csv('wbc_team.csv') wbc_stat<-read.csv('wbc_stat.csv') merged_data<-merge(wbc_team,wbc_stat,by='COUNTRY') print(wbc_team) print(wbc_stat) print(merged_data)	RWbc.txt
Python	import pandas wbc_team = pandas.read_csv('wbc_team.csv') wbc_stat = pandas.read_csv('wbc_stat.csv') merged_data = pandas.merge(wbc_team,wbc_stat,on=['COUNTRY']) print(wbc_team) print(wbc_stat) print(merged_data)	wbc.desiredresult

SAS	libname rosetta 'h:\';
	
	proc import datafile = "wbc_statistics.csv"
	  out=rosetta.wbc_stat dbms=csv replace;
	run;
	
	proc import datafile = "h:\wbc_team.csv"
	  out=rosetta.wbc_team dbms=csv replace;
	run;
	
	proc sort data=rosetta.wbc_stat;
	  by COUNTRY;
	run;
	
	proc sort data=rosetta.wbc_team;
	  by COUNTRY;
	run;
	
	data merged_data;
	  merge rosetta.wbc_stat(in=a) rosetta.wbc_team(in=b);
	  by COUNTRY;
	  if a and b;
	proc print data=merged_data;
	run;
		saswbc.txt

C.6 Convert Numeric to Character String

This section creates a numeric variable named `jack`, checks to see what type of variable it is, converts `jack` into a character string, and confirms that `jack` has been converted into a character string.

Convert Numeric into String	Code	Desired Result
R	jack=17.01 print(jack) print(typeof(jack)) jack_str=toString(jack) print(is.character((jack_str)))	[1] 17.01 [1] "double" [1] TRUE
Python	jack=17.01 print(jack) print(type(jack)) jack_str=(str(jack)) isinstance(jack_str,str)	17.01 <class 'float'> Out[29]: True

SAS	data testing;
	  length cjack $10;
	  jack=17.01;
	  cjack=put(jack,5.2);
	  if jack-int(jack)=0 then v_type="integer";
	  else v_type="float";
	  if vtype(cjack)="C" then flag="true";
	  else flag="false";
	run;
	proc print data=testing;
	  var jack v_type flag;
	run;
		Obs jack v_type flag
		1 17.01 float true

C.7 Concatenate Character Strings Separated by a Comma

Concatenate different character strings into one-character string with a comma separator.

Concatenate Strings	Code	Desired Result
R	mystring<-"CDARs of Lebanon" cat(mystring,"tall and lofty",sep=", ")	CDARs of Lebanon, tall and lofty
Python	mystring = "CDARs of Lebanon" ", ".join([mystring,"tall and lofty"])	'CDARs of Lebanon, tall and lofty'

SAS	data concat;
	  mystring="CDARs of Lebanon";
	new3=catx(', ',mystring,"tall and lofty");
	run;
	proc print data=concat;
	  var new3;
	run;
		Obs new3
		1 CDARs of Lebanon, tall and lofty

C.8 Concatenate Numbers Separated by a Space

Concatenate different character strings into one-character string separated by a space.

Concatenate Numbers Separated by a Space	Code	Desired Result
R	cat(1:10)	1 2 3 4 5 6 7 8 9 10
Python	" ".join([str(i) for i in range(1,11)])	'1 2 3 4 5 6 7 8 9 10'

SAS	data catnum;
	  /* A string of numbers without the separator. */
	  newstring=catx(' ',1,2,3,4,5,6,7,8,9,10);
	proc print data=catnum;
	run;
		Obs newstring
		1 1 2 3 4 5 6 7 8 9 10

C.9 Round Numbers

In this module, we take an array of numbers called dez, and round them in three ways: (1) take the *floor*, which is the largest integer equal or smaller than the number, the *ceiling*, which is the smallest integer equal or greater than the number, and the *round*, the nearest integer to the number.

Round Numbers	Code	Desired Result
R	dez<-c(4.6,3.6,4.2,1.9,56.8,12.0,1.8) print(dez) print(floor(dez)) print(ceiling(dez)) print(round(dez))	[1] 4.6 3.6 4.2 1.9 56.8 12.0 1.8 [1] 4 3 4 1 56 12 1 [1] 5 4 5 2 57 12 2 [1] 5 4 4 2 57 12 2
Python	import numpy dez = numpy.array([4.6,3.6,4.2,1.9,56.8,12.0,1.8]) print(dez) print(numpy.floor(dez)) print(numpy.ceil(dez)) print(numpy.round(dez))	[4.6 3.6 4.2 1.9 56.8 12. 1.8] [4. 3. 4. 1. 56. 12. 1.] [5. 4. 5. 2. 57. 12. 2.] [5. 4. 4. 2. 57. 12. 2.]

SAS	proc iml;
	dez={4.6 3.6 4.2 1.9 56.8 12.0 1.8};
	fdez=floor(dez);
	cdez=ceil(dez);
	rdez=round(dez);
	m=dez//fdez//cdez//rdez;
	print(m);
	quit;
		m
		4.6 3.6 4.2 1.9 56.8 12 1.8
		4 3 4 1 56 12 1
		5 4 5 2 57 12 2
		5 4 4 2 57 12 2

C.10 Format Numbers to Four Significant Digits

In this module, we take the number π and format it to four significant digits the following ways: with one leading space, in scientific notation, to four significant figures, and with leading negative and positive signs. The last three of the four use C++ formatting for R and Python, and both languages are based on C. *Sprintf* stands for "string print."

All numbers will be rounded to four significant digits, in accordance with Data Stewardship and Executive Policy Committee (DSEP) rounding policy of the U.S. Census Bureau.

C.10.1 Produce Designated Number of Significant Digits Using C++ Style Formatting

Format to Significant Digits	Code	Desired Result
R	print(sprintf("% 4g", pi))	[1] "3.142"
Python	from math import pi print(format(pi,'.4g'))	3.142

SAS	data sigdig;
	w=constant("pi");
	w1=round(w,10**(int(log10(abs(w)))-3));
	run;
	
	proc print data=sigdig;
	var w1;
	run;
		Obs **w1**
		1 3.142

C.10.2 Produce Numbers with Leading Spaces

Format with Leading Space	Code	Desired Result
R	print(sprintf("% .4g",pi))	[1] " 3.142"
Python	from math import pi print("{:6.3f}".format(pi,'.4g'))	3.142

SAS	data lead;
	y=constant("pi");
	y_round=round(y,10**(int(log10(abs(y)))-3));
	y1=" "||y_round;
	run;
	
	proc print data=lead;
	var y_round y1;
	run;
		Obs y_round y1
		1 3.142 3.142

C.10.3 Produce Numbers in Scientific Notation Using C++ Style Formatting

The authors are not aware of a method to both simultaneously force the number into scientific notation *and* format to a certain number of significant figures.

Produce Numbers in Scientific Notation	Code	Desired Result
R	print(sprintf("%e",pi)) #Exponential Notation with lower-case "e" print(sprintf("%E",pi)) #Exponential Notation with upper-case "e"	[1] "3.141593e+00" [1] "3.141593E+00"
Python	from math import pi print("%e" % pi) print("%E" % pi)	3.141593e+00 3.14E+00
SAS	data scinote; z=constant(“pi”); run; proc print data=scinote; format z E13.6; var z; run;	**Obs z** 1 3.141593E+00

C.10.4 Produce Numbers with Positive and Negative Signs Using C++ Style Formatting

Positive and Negative Signs	Code	Desired Result
R	sprintf("%+.4g",pi) sprintf("%+.4g",-pi)	[1] "+3.142" [1] "-3.142"
Python	from math import pi print("%+.4g" % pi) print("%+.4g" % -pi)	3.142 -3.142

SAS	data signage;
	a=constant("pi");
	a_round=round(a,10**(int(log10(abs(a)))-3));
	b=-a_round;
	run;
	
	proc print data=signage;
	var a_round b;
	run;
		Obs a_round b
		1 3.142 -3.142

C.11 Create a Function

In this module, we create a function, $expy = a \times 3^b$, wherein the user can input his own values for a and b. In SAS, this is done in a macro. The executable part of an R function is enclosed by curly-braces, while that in Python is blocked off by indentation.

Create a Function to Take the Exponent, with User-defined Parameters	Code	Desired Result
R	expy = function(a,b) {a*exp(0.5*b)} print(expy(3,7))	[1] 99.34636
Python	from math import exp def expy(a,b): return a*exp(0.5*b) print(expy(3,7))	99.3463558761

SAS	%macro expy(a,b);
	data expy;
	expy=&a*exp(0.5*&b);
	proc print data=expy;
	run;
	%mend;
	%expy(3,7);
	run;
		Obs expy
		1 99.3464

C.12 Impute Missing Values and Collapse Variables

Imputing missing values is a large part of the work done at the U.S. Census Bureau. We create a vector with missing values (denoted `NA`, `NaN`, and . in R, Python, and SAS, respectively). Then we use an if-else statement to replace missing values with zeroes. The `ifelse` function in R works similar to the =IF statement in Excel—the arguments, are the condition, the value if the condition is true, and then value if the condition is false. In Python, `where` replaces `ifelse`. In SAS, the extended form of an if-else statement is used. Section C.11.3 entails taking the same vector we have created, and changing all numbers 5 and under to 5, and all numbers over 5 to 10.

C.12.1 Create a Vector with Missing Values

Create a Vector with Missing Values	
R	**Code**

	#Ifelse and Nested Ifelse Statement, Similar to Excel's =IF()
	flow<-c(1,2,3,NA,6,7,9,NA) #A vector with the 4th&10th elements missing
	print(flow)

	Desired Result
	[1] 1 2 3 NA 6 7 9 NA
Python	**Code**
	import numpy flow=numpy.array([1,2,3,None,6,7,9,None]) print(flow)
	Desired Result
	[1 2 3 None 6 7 9 None]
SAS	**Code**
	data flow_master; array flow {8} flow1-flow8; input (flow1-flow8) (: 1.); datalines;
	Desired Result
	Obs flow1 flow2 flow3 flow4 flow5 flow6 flow7 flow8 **1** 1 2 3 . 6 7 9 .

C.12.2 Impute Missing Values of a Vector Using Conditional Processing

<table>
<tr><th colspan="2">Impute Missing Values</th></tr>
<tr><td rowspan="3">R</td><td>Code</td></tr>
<tr><td>flow.imputed<-ifelse(is.na(flow)==TRUE,0,flow)
print(flow.imputed)</td></tr>
<tr><td>Desired Result</td></tr>
<tr><td></td><td>[1] 1 2 3 0 6 7 9 0</td></tr>
<tr><td rowspan="4">Python</td><td>Code</td></tr>
<tr><td>flow_imputed = flow
flow_imputed[numpy.where(numpy.equal(flow,None))]=0
print(flow_imputed)</td></tr>
<tr><td>Desired Result</td></tr>
<tr><td>[1 2 3 0 6 7 9 0]</td></tr>
<tr><td rowspan="4">SAS</td><td>Code</td></tr>
<tr><td>data flow_imputed;
  set flow_master;
  array flow {8} flow1-flow8;
  do k=1 to 8;
  if flow[k]=. then flow[k]=0;
  end;
run;

proc print data=flow_imputed;
  var flow1-flow8;
run;</td></tr>
<tr><td>Desired Result</td></tr>
<tr><td>Obs flow1 flow2 flow3 flow4 flow5 flow6 flow7 flow8
1 1 2 3 0 6 7 9 0</td></tr>
</table>

C.12.3 Collapse a Variable

Collapse Variable	
R	**Code**
	flow.imp.collapsed<-ifelse(is.na(flow)==TRUE,0,ifelse(flow>5,10,5)) print(flow.imp.collapsed)
	Desired Result
	[1] 5 5 5 0 10 10 10 0
Python	**Code**
	flow_imp_collapsed = numpy.array([10 if i>5 else 5 for i in flow_imputed]) print(flow_imp_collapsed)
	Desired Result
	[5 5 5 0 10 10 10 0]
SAS	**Code**

	data flow_collapsed;
	set flow_master;
	array flow {8} flow1-flow8;
	do k=1 to 8;
	if flow[k]=. then flow[k]=0;
	else if flow[k] > 5 then flow[k]=10;
	else flow[k]=5;
	end;
	run;
	
	proc print data=flow_collapsed;
	var flow1-flow8;
	run;

	Desired Result
	Obs flow1 flow2 flow3 flow4 flow5 flow6 flow7 flow8 **1** 5 5 5 0 10 10 10 0

Appendix D: Beginning Statistics (for new statisticians)

D.1 Compute Frequencies

In this module, a football CSV file named Poly.csv is read into a data frame, then bivariate frequencies are computed. Note that in SAS, if the user outputs as a table, absolute frequencies are computed; however, if the output is to a list, the user does not see this.

Compute Frequencies	Code	Desired Result
R	poly<-read.csv("poly.csv") polytab<-table(poly$Status,poly$Heritage) print(polytab)	Non-Polynesian Polynesian Missed 4 21 Signed 8 9
Python	import pandas poly=pd.read_csv("poly.csv") polytab=pd.crosstab(poly.Status,poly.Heritage) print(polytab)	Heritage Non-Polynesian Polynesian Status Missed 4 21 Signed 8 9

SAS	proc import datafile="h:/poly.csv" out=poly dbms=csv replace;
	
	proc freq data=poly;
	tables Status*Heritage/norow nocol nopercent nocum list;
	run;
		Status Heritage Frequency
		Missed Non-Polynesian 4
		Missed Polynesian 21
		Signed Non-Polynesian 8
		Signed Polynesian 9

D.2 Compute Total Proportional Frequencies

This module entails computing proportional bivariate frequencies. Also note that proportions in R and Python are represented by decimals, while in SAS they are represented as percentages. For instance, 0.0952 in R is 9.52 [%] in SAS.

Compute Total Proportional Frequencies

R

Code

```
poly<-read.csv("poly.csv")
polytab<-prop.table(poly$Status,poly$Heritage)
pptab<-prop.table(polytab)
pptab.df<-data.frame(unclass(pptab))
status<-rownames(polytab)
pptab.df.fmt<-data.frame(lapply(pptab.df,sprintf,fmt="%.4g"))
Status<cbind(status,pptab.df.fmt)
print(Status)
```

Desired Result

```
  Status Non.Polynesian Polynesian
1 Missed        0.09524        0.5
2 Signed         0.1905     0.2143
```

Python

Code

```
import os

import pandas as pd
os.chdir('c:/Users/Owner/')
pd.options.display.float_format="{0:1.4g}".format
poly = pd.read_csv("poly.csv")
polytab=pd.crosstab(poly.Status,poly.Heritage,normalize='all')
```

Desired Result

```
print(polytab)
Heritage  Non-Polynesian  Polynesian
Status
Missed          0.09524        0.5
Signed           0.1905      0.2143
```

SAS

Code

```
proc import datafile="h:/poly.csv" out=poly dbms=csv replace;

proc freq data=poly;
 tables Status*Heritage /norow nocol nocum list;
run;
```

Desired Result

Status	Heritage	Frequency	Percent
Missed	Non-Polynesian	4	9.52
Missed	Polynesian	21	50.00
Signed	Non-Polynesian	8	19.05
Signed	Polynesian	9	21.43

D.3 Compute Row Proportional Frequencies

The following code computes proportional frequencies for each row. R uses 1 to designate row and 2 to designate column. Python generally uses 0 for row and 1 for column, but in this case, it's "index" for row and "columns" for column. When computing proportional frequencies across rows or columns in SAS, the user cannot use the `list` option. However, for tables in SAS, the engine automatically computes absolute frequencies; in R and Python, this does not happen. Also in SAS, one would specify `nocol` to specify that we want column frequencies only. Note that SAS contains the absolute frequencies in the rows above the proportional frequencies for tables. As before, proportions are represented as decimals in R and Python, but as percentages in SAS.

Compute Row Proportional Frequencies

R

Code

```
poly<-read.csv("poly.csv")
polytab<-prop.table(poly$Status,poly$Heritage)
pptab<-prop.table(polytab,1)
pptab.df<-data.frame(unclass(pptab))
status<-rownames(polytab)
pptab.df.fmt<-data.frame(lapply(pptab.df,sprintf,fmt="%.4g"))
Status<cbind(status,pptab.df.fmt)
print(Status)
```

Desired Result

```
      Status Non.Polynesian Polynesian
1 Missed         0.16       0.84
2 Signed        0.4706      0.5294
```

Python

Code

```
import os
import pandas as pd
os.chdir('c:/Users/Owner/')
pd.options.display.float_format="{0:1.4g}".format
poly = pd.read_csv("poly.csv")
polytab=pd.crosstab(poly.Status,poly.Heritage,normalize='index')
print(polytab)
```

Desired Result

```
Heritage  Non-Polynesian  Polynesian
Status
Missed            0.16       0.84
Signed           0.4706      0.5294
```

SAS

Code

```
proc import datafile="h:/poly.csv" out=poly dbms=csv replace;
proc freq data=poly;
 tables Status*Heritage/nocol;
run;
```

Desired Result

Table of Status by Heritage

	Status	Heritage		
		Non-Polynesian	Polynesian	Total
Frequency	Missed	4	21	25
Percent		9.52	50.00	59.52
Col Percent		16.00	84.00	
	Signed	8	9	17
		19.05	21.43	40.48
		47.06	52.94	
	Total	12	30	42
		28.57	71.43	100.00

D.4 Compute Column Proportional Frequencies

The directions for computing column percentage frequencies are spelled out in the previous section.

<table>
<tr><th>Compute Column Proportional Frequencies</th><th>Code</th></tr>
<tr><td rowspan="3">R</td><td>poly<-read.csv("Poly.csv")
polytab<-table(poly$Status,poly$Heritage)
print(polytab)
fish<-fisher.test(polytab,alternative="less",conf.int=F)
print(noquote(sprintf('%.4g',fish$p.value)))</td></tr>
<tr><td>Desired Result</td></tr>
<tr><td>Non-Polynesian Polynesian
Missed 4 21
Signed 8 9
[1] 0.03342</td></tr>
<tr><td rowspan="3">Python</td><td>Code</td></tr>
<tr><td>import pandas as pd
import scipy.stats as stats
polytab=pd.crosstab(poly.Status,poly.Heritage)
print(polytab)
oddratio,pvalue=stats.fisher_exact(polytab,alternative='less')
print(format(pvalue,'.4g'))</td></tr>
<tr><td>Desired Result</td></tr>
<tr><td>Heritage Non-Polynesian Polynesian
Status
Missed 0.3333 0.7
Signed 0.6667 0.3</td></tr>
<tr><td rowspan="3">SAS</td><td>Code</td></tr>
<tr><td>proc import datafile=”/poly.csv" out=poly dbms=csv replace;
proc freq data=poly;
tables Status*Heritage/norow;
run;</td></tr>
<tr><td>Desired Result</td></tr>
<tr><td>Table of Status by Heritage

Status Heritage
Non-Polynesian Polynesian Total
Frequency Missed 4 21 25
Percent 9.52 50.00 59.52
Col Percent 33.33 70.00

Signed 8 9 17
19.05 21.43 40.48
66.67 30.00

Total 12 30 42
28.57 71.43 100.00</td></tr>
</table>

Appendix E: Advanced Statistical Modeling (for experienced statisticians and programmers)

E.1 Run Fisher Test

Among the Fisher Test's a Fisher Exact test is one used for small sample sizes. In all three programming languages of interest, running a Fisher test entails creating a frequency table first. In R and Python, creating the table and administering the test are done sequentially. In SAS, use `fisher` in `proc freq`.

Run Fisher Test

R

Code

```
poly<-read.csv("Poly.csv")
polytab<-table(poly$Status,poly$Heritage)
print(polytab)
fish<-fisher.test(polytab,alternative="less",conf.int=F)
                    print(noquote(sprintf('%.4g',fish$p.value)))
```

Desired Result

```
Non-Polynesian Polynesian
  Missed              4         21
  Signed              8          9
[1] 0.03342
```

Python

Code

```
import pandas as pd
import scipy.stats as stats
polytab=pd.crosstab(poly.Status,poly.Heritage)
print(polytab)
oddratio,pvalue=stats.fisher_exact(polytab,alternative='less')
print(format(pvalue,'.4g'))
```

Desired Result

```
Heritage  Non-Polynesian  Polynesian
Status
Missed                4          21
Signed                8           9
0.03342
```

SAS

Code

```
options nocenter pageno=1;
ods pdf file='M:\Rosetta Wiki\SAS\fisher_exact.pdf';
ods pdf startpage=no;

proc import datafile="poly.csv" out=poly dbms=csv replace;
proc freq data=poly;
 tables Status*Heritage /norow nocol nopercent;
run;
proc freq noprint data=poly;
  tables Status*Heritage /chisq;
  output out=poly2(keep=xpl_fish) chisq;
run;
proc print data=poly2;
  title 'Fisher Exact Left-Sided P-value';
run;
```

Desired Result

Table of Status by Heritage

	Status	Heritage		
		Non-Polynesian	Polynesian	Total
Frequency	Missed	4	21	25
	Signed	8	9	17
	Total	12	30	42

Obs	XPL_FISH
1	0.0334

E.2 Add Laplace Noise to Variable

This module entails:

1. Reading in a CSV dataset on the effect drugs have on aging over several species.
2. Finding the expected value of the avg_lifespan_change.
3. Finding out the number of rows (or records) in the data frame.
4. Generating a vector of random number equal to the number of rows in the dataframe with mean zero, and adding it to avg_lifespan_change to create a new variable lsc_noisy
5. Taking the expected value lsc_noisy. It should roughly equal avg_lifespan_change.

Note that when implementing step 3 in SAS, output is to the log. Also note that SAS currently does not have the Laplace distribution built in to the RAND function, so PDF is used.

Add Laplace Noise to a Variable	Code	Desired Result
R	library(rmutil) drugage<-read.csv("drugage.csv") sprintf("%.4g",mean(drugage$avg_lifespan_change,na.rm=T)) print(nrow(drugage)) noise=rlaplace(length(is.na(drugage$avg_lifespan_change==F)),0) drugage$lsc_noisy=drugage$avg_lifespan_change+noise sprintf("%.4g",mean(drugage$lsc_noisy,na.rm=T))	[1] 13.13 [1] 1316 [1] 13.11
Python	import pandas as pd drugage=pd.read_csv("drugage.csv") import numpy as np print(format(drugage["avg_lifespan_change"].mean(),'.4g')) print(len(drugage.index)) noise=np.random.laplace(0,1,len(drugage.index)) drugage["lsc_noisy"]=drugage["avg_lifespan_change"]+noise print(format(drugage["lsc_noisy"].mean(),'.4g'))	13.13 1316 13.17

SAS	proc import datafile="drugage.csv" out=drugage dbms=csv;
	data drugage_rec;
	  set drugage end=eof;
	  nobs=_N_;
	  if (eof) then output;
	run;
	proc print data=drugage_rec;
	  var nobs;
	run;
	data drugage;
	  set drugage;
	  noise = pdf('LAPLACE',1);
	  lsc_noisy=avg_lifespan_change + noise;
	run;
	proc means data=drugage mean noprint;
	  var avg_lifespan_change lsc_noisy;
	  output out=drugage_stat(drop=_TYPE_ _FREQ_);
	run;
	data drugage_stat;
	  set drugage_stat;
	  if _STAT_ = 'MEAN ';
	run;
	proc transpose data=drugage_stat
	out=drugage_tr(rename=(_NAME_=Variable COL1=Mean));
	data drugage_means;
	  set drugage_tr;
	  Mean=round(Mean,10**(int(log10(abs(Mean)))-3));
	run;
	proc print data=drugage_means;
	run;
		Obs **nobs**
		1 1316
		
		Obs **Variable** **Mean**
		1 avg_lifespan_change 13.13
		2 lsc_noisy 13.32

E.3 Add Gaussian Noise to Variable

This module entails:

1. Reading in a CSV dataset on the effect drugs have on aging over several species.
2. Finding the expected value of the avg_lifespan_change.
3. Finding out the number of rows (or records) in the data frame.
4. Generating a vector of random numbers equal to the number of rows in the dataframe with mean zero, and a standard deviation of 5, and adding it to avg_lifespan_change to create a new variable lsc_noisy.
5. Taking the expected value lsc_noisy. It should roughly equal avg_lifespan_change.

Due to differences in handling missing values, R requires modestly more work than Python, which just treats them as if they did not exist. In SAS, step 3 outputs the number of records to the log.

Add Gaussian Noise to a Variable	Code	Desired Result
R	drugage<-read.csv("drugage.csv") sprintf("%.4g",mean(drugage$avg_lifespan_change,na.rm=T)) print(nrow(drugage)) n=length(is.na(drugage$avg_lifespan_change==F)) noise=rnorm(n,0,5) drugage$lsc_noisy=drugage$avg_lifespan_change+noise sprintf("%.4g",mean(drugage$lsc_noisy,na.rm = T))	[1] "13.13" [1] 1316 [1] "13.09"
Python	import pandas as pd drugage=pd.read_csv("drugage.csv") import numpy as np print(drugage["avg_lifespan_change"].mean()) n=print(len(drugage.index)) print(n) noise=np.random.normal(0,5,n) drugage["lsc_noisy"]=drugage["avg_lifespan_change"]+noise print(format(drugage["lsc_noisy"].mean(),'.4g'))	13.13 1316 13.09

SAS	proc import datafile="drugage.csv" out=drugage dbms=csv;
	data drugage_rec;
	  set drugage end=eof;
	  nobs=_N_;
	  if (eof) then output;
	run;
	proc print data=drugage_rec;
	  var nobs;
	run;
	data drugage;
	  set drugage;
	  noise = rand('NORMAL',0,5);
	  lsc_noisy=avg_lifespan_change + noise;
	run;
	proc means data=drugage mean noprint;
	  var avg_lifespan_change lsc_noisy;
	  output out=drugage_stat(drop=_TYPE_ _FREQ_);
	run;
	data drugage_stat;
	
	  set drugage_stat;
	
	if _STAT_ = 'MEAN ';
	run;
	proc transpose data=drugage_stat
	out=drugage_tr(rename=(_NAME_=Variable COL1=Mean));
	data drugage_means;
	 set drugage_tr;
	  Mean=round(Mean,10**(int(log10(abs(Mean)))-3));
	run;
	proc print data=drugage_means;
	run;
		Obs **nobs**
		1 1316
		
		Obs **Variable** **Mean**
		1 avg_lifespan_change 13.13
		2 lsc_noisy 13.32